Simulation-Based Intelligent Robotic Agent for Space Station Freedom

Csaba A. Biegl, James F. Springfield, George E. Cook, *Kenneth R. Fernandez

Vanderbilt University, Dept. of Electrical Engineering,
Box 1824, Sta. B, Nashville, TN 37235
*NASA Marshall Space Flight Center, Huntsville, AL

Abstract

This paper describes a robot control package which utilizes on-line structural simulation of robot manipulators and objects in their workspace. The model-based controller is interfaced with a high-level agent-independent planner, which is responsible for the task-level planning of the robot's actions. Commands received from the agent-independent planner are refined and executed in the simulated workspace, and upon successful completion, they are transferred to the real manipulators.

1. Introduction

It is expected that robotic systems will play a crucial rule in the operation of Space Station Freedom. Various repetitive tasks associated with the maintenance of the Station are the most likely candidates for robotics and automation applications, since such applications would free up a considerable amount of crew-time for more useful activities, like scientific experiments, etc.. Controlling these robots will be a very complex task due to unusual operating conditions, like tight space, strong interdependence of various subsystems onboard the station, and the unavoidable presence of humans near the robot's working area. These requirements will necessitate the development of highly specialized robot control systems for space applications. The goal of the research described in this paper is to design and implement an autonomous robotic agent which can serve as a component of such control systems.

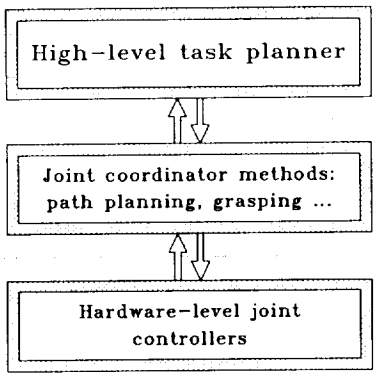

Figure 1. Multilayer Robot Control System Architecture

It is customary to distinguish between three layers in robot control systems, as seen in Figure 1. The bottom layer implements the low-level joint control systems. At this level the controllers' inputs are individual joint position and speed signals, and their output determines the voltage or hydraulic pressure applied to the joint actuators of the manipulator.

The middle layer of a hierarchical robot control system contains the modules which are necessary for the coordinated motion of the individual manipulator joints. Operations typically performed at this level include path synthesis, object grasping, and compliant motion.

The components of the highest layer implement the task planning functionalities in the system. Task planning means breaking down a complex goal into a sequence of simple actions suitable for input at the path synthesis level.

An agent-independent high-level task planner has been described in [3]. The goal of the work described in this paper was to provide a target environment for this task planner, which is capable of executing the planner's action sequences using either a graphics simulation environment or real robot hardware. The operation of this robotic agent is based on the structural, geometrical modeling of the robot manipulators and the objects in their workspace. The reason for this is that the techniques used to model solid objects provide the most natural way to describe the robots and the workspace objects. Solid modeling techniques might be used in different areas of the design and operation of robotics systems, like:

- *Design and testing of manipulators:* In such applications the purpose of the modeling is to study different approaches to fulfill the design specifications of the manipulator.

- *Robot action planning:* The modeling environment is used to build a representation of the robot and the objects in the workspace, and to create and validate action plans by simulating various actions in the model space.

- *On-line control of robot manipulators:* The modeling tools are integrated into a hierarchical robot control system, and the action plans generated and tested in the model space will be transmitted to the robot manipulators for execution.

2. Related Work -- A Robot Modeling Environment

The Intelligent Robotic Agent is based on a robot simulation and control environment described in [2]. The main requirements against a geometrical modeling toolkit used in robotics applications can be summarized as follows [1][4][7]:

- *A set of solid primitives* (like boxes, cylinders, cones, etc...) which can be used to construct the manipulator and world models.

- *A way to build structured objects* from the primitives or other previously defined structured types.

- *A set of methods to manipulate the models.* These include methods for accessing and animating objects, collision detection algorithms, and routines for forward and inverse kinematics calculations, graphics display, and possibly dynamics calculation.

These goals were satisfied by implementing a robot simulation library. The basis for this development was the ROBOSIM [5] robot simulation package. ROBOSIM in its original form is a command line oriented graphical robot modeling package. It provides commands to build the geometrical representation of robots and various objects and to perform simple operations on these.

To enhance the object manipulation capabilities of the original ROBOSIM package, a

simulation library has been built which is capable of operating on the internal geometrical model databases built up by ROBOSIM's command interpreter. This library includes the following components:

- *The animation module* provides the methods to "operate" the manipulators in the model space. Its methods include routines for driving the arms' joints, straight line motion, grasping, etc...

- *The inverse kinematics module* aids the animation by providing the inverse kinematics solutions for the robot arm(s) in the model space. By default it uses a numerical method, but if the analytical inverse equations are available, the default method can be replaced by these.

- *The collision detection module* is used by the animation module to check the validity of the steps during a movement. If the step would result in a collision, a report is sent to the animation module, describing the objects involved in the collision.

- *The robot interface driver module* is used to duplicate the actions taken in the simulation environment in the real robot's workspace. It generates calls to a low-level interface library based on the simulated joint movements. Error reporting is also possible in case the command fails in the real workspace.

- *The graphics display module* is used to display the workspace configuration and to follow the actions in the simulated environment.

The structure of the modeling environment composed of the above modules and the original ROBOSIM interpreter can be seen in Figure 2.

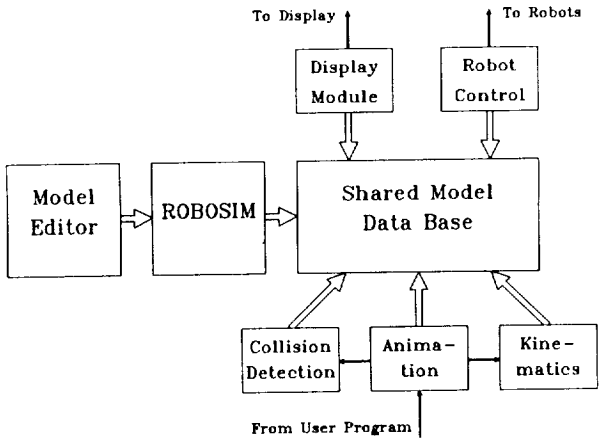

Figure 2. Components of the Robot Modeling Environment

3. The Intelligent Robotic Agent

The purpose of the Intelligent Robotic Agent is to provide a user-friendly way to use the tools of the Robot Modeling Environment described above. The agent was designed to be able to receive robot action sequences either from a high-level symbolic task planner or

directly from the user during an interactive session. The main requirements against the agent are summarized below:

- *Symbolic Planner/User Interface Protocol:* Regardless of the operating mode of the agent (used by a task planner or by an interactive user), the same command interface is being used. This command language provides symbolic manipulator and object identifiers for a more user-friendly programming environment. The interface protocol also includes extensive error reporting mechanisms.

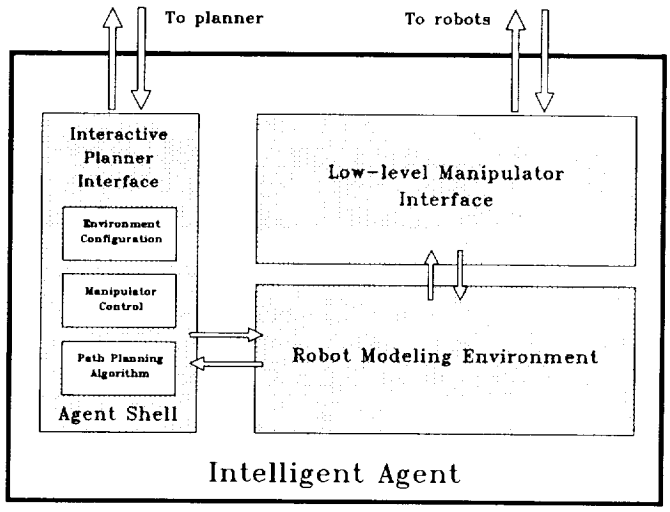

Figure 3. Intelligent Agent System Architecture

- *Environment Configuration Interface:* In order to ensure the consistency of the world models with the task planner, the agent includes a set of commands forming a configuration interface which is used by the planner (or by the user in an interactive session) to build the geometrical representation of the workspace. The agent features its own modeling language for creating elementary objects, building complex objects from these, and to perform various transformations. In view of the considerable effort already invested in creating models of various objects and robots in the ROBOSIM language, the possibility of using existing ROBOSIM models is also provided.

- *Agent Operation Interface:* A set of commands is provided to operate the robot manipulator models in the agent's data base. These include commands for various types of motion (joint interpolated, straight line, etc..), path planning with collision avoidance, and for grasping and releasing objects.

- *Graphics Display:* Probably one of the most useful features of any robot simulation package is the possibility to view the manipulators in their workspace as they perform various actions, in a safe, simulated environment. The agent includes a graphics display feature which is usable during both the environment configuration and the agent operation phases.

- *Interface to Robot Controllers:* The agent can operate in either of two modes. In the first mode the actions received from the command interface are performed only in the geometrical modeling environment. In this case the execution can be viewed on a graphics display, and the agent will report any error conditions encountered during the execution (collisions, joint violations, etc..). In the second mode the successful execu-

tion of the above steps is followed by sending commands to a real robot hardware interfaced to the agent to duplicate the actions taken in the simulated environment in the real workspace too.

The above goals were satisfied by incorporating the services of the modeling environment into an interactive shell, and by creating a low-level interface package between the simulation library's robot interface driver module and the controller of the manipulator used in the system. The architecture of the Intelligent Agent can be seen in Figure 3.

The agent shell's task is to interpret the commands sent by the planner. It has a built-in symbol table for storing the various objects in the workspace and the attributes associated with these. After parsing the commands, the shell will invoke the appropriate functions of the robot simulation library with an argument structure built using the information stored in the symbol table. Any results or error codes returned by the library call are translated into the corresponding message format and sent back to the planner, or printed to the user during an interactive session.

One important design consideration associated with the agent was the minimization of the numerical calculations required from the task planner. To accomplish this goal various attributes are associated with each object. These attributes include the object's current position, predefined grasping points, and predefined joint parameter vectors for the robot manipulator objects. These attributes are linked to the object, i.e. any transformation will update their values. This means, that for example the task planner does not have to compute numerical coordinates to move a robot manipulator to an object's grasping point, but it can use the symbolic grasping point attribute whose value is always linked to the object's current position.

The agent shell also includes a path planning algorithm, which is used to synthesize a collision-free path between two locations in the workspace. This algorithm relies on the collision detection feature of the simulation library and utilizes a simple search strategy to find an alternate route in case a collision is detected during a motion segment. First the endpoint of the failed motion segment is checked to see if there is any collision when the manipulator reaches that point. If this is the case then the search fails, since the motion segment can not be completed regardless of the route taken. Otherwise the search algorithm will subdivide the failed motion segment by inserting an intermediate point selected using a set of heuristic rules, and recursively repeat the above process. There may be situations when this method will fail to find a path even if such a path exists, but the algorithm described above has been observed to find a solution in the majority of the situations examined so far.

The agent shell also supports the parallel operation of robot manipulators. An extended command line format makes it possible to enter commands for every manipulator in the system at the same time, and these commands will execute parallelly.

4. An Application Example

The Intelligent Agent has been used as an interface between a robotic task planner and a PUMA 560 manipulator. The architecture of the whole system can be seen in Figure 4. The low-level robot interface between the agent shell and the PUMA manipulator's controller has been implemented in a distributed fashion, partly on the workstation running the agent shell, and partly on an IBM PC AT. The reason for this solution was that parts of this interface package have to meet certain real-time requirements.

The configuration described above has been used to execute task plans for repairing electronic equipment. The operations included setting various switches and dials on the front panels of the racks housing the equipment, and replacing faulty circuit boards. A graphics display of the simulated workspace of this application can be seen in Figure 5.

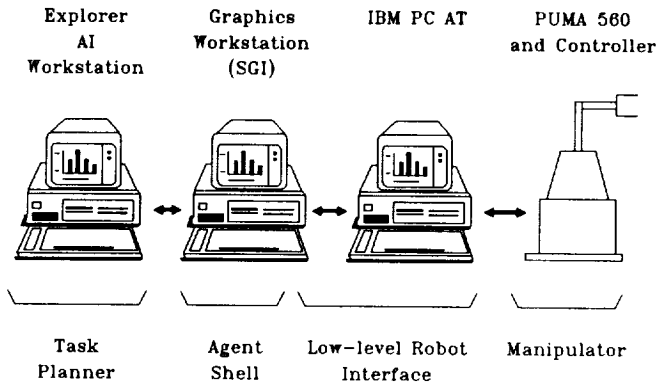

Figure 4. System Configuration for Intelligent Agent Application Example

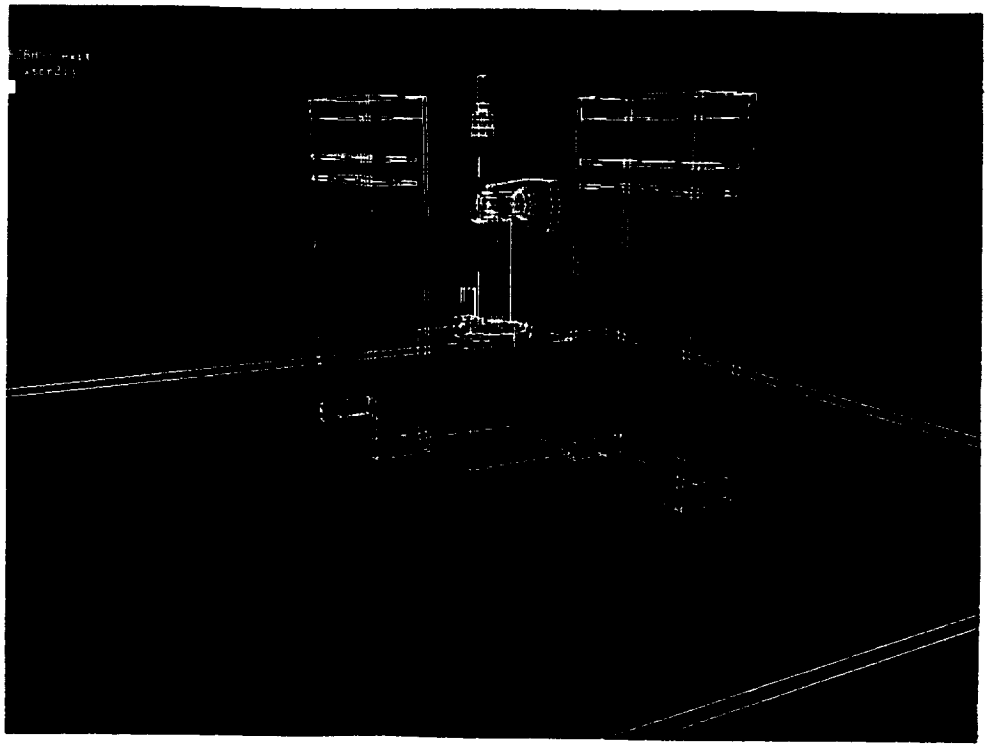

Figure 5. Workspace Configuration for Intelligent Agent Application

5. Summary and Future Research

An intelligent agent has been developed for the execution of robotic task plans generated by a high-level planner. The agent uses a well defined protocol for receiving environment configuration and manipulator commands from the planner and to send back status reports. After translation, the agent uses the services of a robot simulation and control environment to carry out the high-level commands of the planner. This configuration has been used to examine the possibilities of using robots for repairing electronic equipment.

The advantage of this approach is its flexibility. It is easy to adapt the system to different workspace configurations or to different robot manipulators by changing the environment configuration commands sent to the agent. The well-defined planner-agent interface simplifies the development and verification of new task planners, because the task plans can be tested for validity by the agent, and the user can also see the effects of the planner's output on the graphics display.

The most important area of planned future research is the upgrading of the low-level robot manipulator interface. The current interface uses the PUMA manipulator's command language (VAL II [8]) to execute the robot movement commands, but a new version is currently being developed which will bypass this command language and use the PUMA controller's "real-time path control" feature to achieve an even better quality of joint control. This new version will be built using the so called Multigraph Architecture [6], which provides a dynamically configurable distributed macro-dataflow computational environment. This will allow the selection of the joint control scheme and its parameters from the agent shell, resulting in a highly optimized controller performance for every application.

6. Acknowledgements

William S. Davis and Ray Carnes (Boeing Aerospace Company, Huntsville, AL), and Professor Janos Sztipanovits (Vanderbilt University, Nashville, TN) provided valuable technical information and criticism during the development of this work. Some of the equipment necessary for the work was provided by the Hewlett Packard Company and the Boeing Aerospace Company.

7. REFERENCES

1. B. Bhanu, C. C. Ho: "CAD-Based 3D Object Representation for Robot Vision", *Computer,* Vol. 20, No. 8, Aug. 1987, pp. 19-35

2. C. A. Biegl, J. Springfield, et. al.: "Adaptive Control of a Dual-Arm Robot Manipulator Using On-Line Graphical Simulation", in *Proceedings of the ISA Forth Annual Workshop on Robotics and Expert Systems*, (August 1989, Palo Alto, CA) vol. 4, pp. 253-259.

3. W. S. Davis: "Robotic Task Planning: Independent of Agents but Dependent on Time", in *Proceedings of the 1989 IEEE Conference on Robotics and Automation,* (May 1989, Scottsdale, AZ) vol. 2, pp. 690-695.

4. E. Dombre, P. Borrel, A. Liegeois: "A CAD System for Programming and Simulating Robots' Actions", in *Computing Techniques for Robots,* ed I. Aleksander, Chapman and Hall, New York, 1985, pp. 222-247.

5. K. R. Fernandez: "Robotic Simulation and a Method for Jacobian Control of a Redundant Mechanism with Imbedded Constraints", Ph.D. thesis, Vanderbilt University, Spring 1988

6. G. Karsai, et. al.: "Knowledge-Based Approach to Real-Time Supervisory Control", *Proc. of the 1988 American Control Conference*, Atlanta, GA, pp. 620-626, 1988.

7. C. Mirolo, E. Pagello: "A Solid Modeling System for Robot Action Planning", *IEEE Computer Graphics & Applications,* January 1989, pp. 55-69

8. Unimation Inc.: "User's Guide to VAL II", Danbury, CT 1986